IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

User Interface for Integrated Spreadsheets and Word Processing Tables

Inventor(s):
Matthew Kotler
Vinod Anantharaman
Chris Franklin
Oliver Fisher
Alexander Gounares
Matthew Morgan
Richard Wolf

ATTORNEY'S DOCKET NO. MS1-580US

TECHNICAL FIELD

This invention relates to computer programs, and particularly, to word processing and spreadsheet programs. More particularly, this invention pertains to an architecture for integrating spreadsheets and word processing tables in HTML (hypertext markup language) documents, and a user interface for the integrated table/spreadsheet.

BACKGROUND

Word processing and spreadsheet programs are two well-known and widely used software applications. Word processing programs permit users to draft letters, write books, and create other word-centric documents on a computer. Word processing programs are typically designed with the author in mind by offering tools and user interfaces that make writing easier, such as edit functions (e.g., cut, copy, paste, find, replace, etc.), spell and grammar checking, document formatting, and the like. Examples of word processing programs include "Word" from Microsoft Corporation and "WordPerfect" from Corel Corporation.

Spreadsheet programs enable users to create financial records, accounting spreadsheets, budgets, and other number-centric documents on a computer. Spreadsheet programs are developed with the accountant in mind, focusing on tools and user interfaces that simplify data entry and data manipulation. Spreadsheets typically offer such functionality as in-cell formulas, automatic recalculation as data changes, multi-sheet referencing, cell formatting according to data type (e.g., dates, currency, percentages, etc.), and the like. One example of a spreadsheet program is the "Excel" application from Microsoft Corporation.

In the past, computer users who wanted to create primarily word-based documents would select a word processing program, while users who wished to produce number-oriented documents turned to spreadsheet programs. In some situations, however, word processing users might need to include numbers and a spreadsheet "look" to an otherwise word-dominated document.

To accommodate such crossover situations, word processing programs evolved to offer tables, a visual structure that could be used to hold and organize numbers and other types of data. Tables arrange data in columns and rows, thereby emulating the spreadsheet "look". Word processing users can insert a table, modify its layout, and change cell formats to achieve a specific visual appearance to their data. Some tables even support rudimentary functions, such as adding a set of contiguous cells. However, these functions do not automatically recalculate. Accordingly, while visually similar to spreadsheets, word processing tables do not support full spreadsheet functionality.

More recently, object-oriented programming and OLE technologies have been used to provide a richer integration experience. With OLE, word processing users who want greater functionality can embed spreadsheet objects into their word processing documents, instead of tables. Essentially, this is akin to embedding an "Excel" spreadsheet (or other spreadsheet program) into a document running on the "Word" program (or other word processing program). The embedded object carries sufficient functionality to allow the user to enter formulas, format cells, recalculate functions, and do all of the things he/she would normally be able to do on a spreadsheet program.

Though the embedded spreadsheet visually resembles a table and provides the desired spreadsheet functionality, it logistically remains a separate program

that must be invoked by the user. OLE requires that both types of application programs—a word processor and a spreadsheet—be installed on the computer. When the user wants to update the embedded spreadsheet, the user invokes the spreadsheet object by moving a mouse pointer to anywhere on the embedded object and double clicking the left mouse button (or via some other actuation mechanism). In response, an instance of the spreadsheet program is executed and the spreadsheet changes appearance from a "table look" to a reduced size spreadsheet program with numbered rows and lettered columns and program specific menus. In this state, the user can change functions, modify data, reformat the spreadsheet, and perform other spreadsheet tasks. When the user is finished, the user returns focus to the word processing document by moving the mouse pointer outside the spreadsheet object and single clicking the left mouse button.

While the OLE approach offers the full spreadsheet functionality within a word processing document, the process is somewhat sophisticated and typically performed by experienced users who are familiar with both spreadsheets and word processing programs. For novice or less experienced users, it may be confusing to see a table and not appreciate the difference between a word processing table and a full-functioning embedded spreadsheet object. From the user standpoint, different operations are used depending upon whether the visible structure is a table or a spreadsheet. Furthermore, common services such as text formatting, spell checking, and the like do not "tunnel" into the embedded OLE objects and thus, the user is forced to run such services for both the document and the embedded spreadsheet.

Thus, even though the final appearance may be visually similar, word processing tables and spreadsheets provide two completely separate mechanisms

for displaying information. Accordingly, there remains a need for better integration of spreadsheet functionality into word processing tables.

With the rapidly growing popularity of the Internet, many documents delivered to and rendered on computers are written in markup languages, such as HTML (hypertext markup language). Markup languages can allow authors to easily construct a desired visual layout of the document. Some HTML documents provide tables that look and function as if they were integrated with the surrounding text. For instance, financial Websites commonly offer informative discussions on retirement planning, college savings, or buying a house and include with those discussions one or more tables that invite the user to fill in their personal financial information and goals. When the user finishes entering the data fields, the document appears to make on-the-fly calculations and present the results together with the discussions.

Despite the appearance of in-document calculations, the HTML document is nothing more than an electronic form that receives data entered by the user. When the user completes entry, the HTML document is submitted to a Web server that extracts the user data and makes the appropriate financial calculations. The server places the results in another HTML document and serves the document back to the user's computer. The submit and reply occur very quickly, so the user may be unaware that the HTML document holding the results is different than the HTML document into which he/she initially entered data. In any event, the traditional separation between spreadsheets and tables has persisted into the Web-based era.

SUMMARY

A system architecture integrates spreadsheet functionality into tables commonly used in word processing programs and HTML documents. The architecture presents a table user interface (UI) that appears a part of the document, and may be surrounded by text and other document elements. In an HTML document, for example, the table is an HTML element constructed along with other elements and rendered together as an integrated document. Once rendered, the table UI visually resembles a table in a non-editing mode and a spreadsheet in an editing mode. The feel of the table, however, remains much like a word processing table in that a user can type multiple paragraphs, create lists, split cells, and so forth. However, unlike typical word processing tables, the table supports full spreadsheet functionality.

Underlying the table UI, one implementation of the architecture separates data handling functions from presentation functions. The architecture includes a table appearance manager to manage how the table appears in a document including such characteristics as table resizing, selection, cut, copy, paste, split, merge, table formatting and so on. The architecture also has a spreadsheet functionality manager to manage the spreadsheet functions for the table, such as recalculation, formula handling, sorting, referencing, and the like.

The bifurcated architecture supports cross-table referencing in which a cell in one table can reference a cell in another table in the same document, even though the tables are separate from one another. As part of the cross-table referencing, the architecture allows a user to reference the cell in the other table using a reference edit operation (e.g., move pointer to cell and click to capture content in the cell). The architecture further accommodates automatic universal

recalculation throughout all tables in the document. Thus, when a user modifies the contents of one table, the architecture automatically recalculates any formulas in any tables affected by the modification.

The architecture also supports nested table structures in which one table is nested within a cell of another table. Many other architectural features and UI features are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of an exemplary architecture for integrating spreadsheet functionality into word processing tables.

Fig. 2 illustrates a screen display of a rendered document having a single table that is capable of spreadsheet functionality. In Fig. 2, the table exhibits a "table look" during a non-editing mode.

Fig. 3 illustrates a screen display of the rendered document, where the table exhibits a "spreadsheet look" during an editing mode.

Fig. 4 is a block diagram of another exemplary architecture for integrating spreadsheet functionality into word processing tables. The architecture of Fig. 4 illustrates an extension of the Fig. 1 architecture by supporting multiple tables and a free floating field.

Fig. 5 illustrates a screen display of a rendered document having multiple tables. In particular, Fig. 5 shows nested tables, where one table is inserted into a cell of another table, and the ability to reference from one table to the other table.

Fig. 6 illustrates a screen display of a rendered document having multiple tables and a free floating field that appear in an edit mode. Fig. 6 demonstrates cross-table referencing, and edit referencing from a free floating.

Fig. 7 is a block diagram of an exemplary computer that implements the architectures of Figs. 1 and 4.

Fig. 8 is a flow diagram of a process implemented by the architectures of Figs. 1 and 4.

Fig. 9 is a diagrammatic illustration of how a user interface table in a rendered document and underlying functional components in the architecture work together during a recalculation operation.

Fig. 10 is a diagrammatic illustration of how multiple UI tables and underlying functional components in the architecture work together during a cross-table reference edit operation.

DETAILED DESCRIPTION

This disclosure describes an architecture that integrates spreadsheet functionality into tables commonly used in word processing programs and HTML documents. The architecture provides a single mechanism for users to combine the best features of a word processing table with the best features of a spreadsheet engine.

In the described implementation, the architecture provides the integrated table and spreadsheet in a document written in a markup language (e.g., HTML). In this manner, the user is afforded the rich HTML formatting options of both text and tables, including table layout changes (e.g., merging and splitting cells), as well as the data specific calculation and formatting features that are traditionally associated only with a separate spreadsheet application. However, it is noted that the architecture may be useful in other document types that are not rooted in a markup language.

Architecture

Fig. 1 shows the architecture 100 that integrates spreadsheet functionality into word processing tables. The architecture 100 may be implemented on a standalone computer, a network server computer, a network client computer, or distributed at both the server and client. The architecture 100 includes a document renderer 102, a table object 104, spreadsheet objects 106, a spreadsheet editor 108, a workbook 110, a spreadsheet engine 112, and one or more non-core worksheet functions 114(1)-114(W) that may be optionally used by the spreadsheet engine 112.

The architecture 100 separates data handling functions from presentation functions of the integrated table/spreadsheet. In this manner, the architecture may be characterized as a cooperation of two system managers: a table appearance manager 116 and a spreadsheet functionality manager 118. The table appearance manager 116 manages how the table appears in a document and facilitates such tasks as table resizing, selection, cut, copy, paste, split, merge, table formatting and so on. The table appearance manager 116 includes the table object 104, the spreadsheet objects 106, and the spreadsheet editor 108. The spreadsheet functionality manager 118 manages the spreadsheet functions for the table, such as recalculation, formula handling, sorting, referencing, and the like. The spreadsheet functionality manager 118 includes the spreadsheet engine 112 and worksheet functions 114. With the bifurcated architecture, the spreadsheet functionality manager 118 is not concerned with the table layout or other visual features, and the table appearance manager 116 is not concerned with data management, formulas, and recalculation processes.

The bifurcated architecture 100 is advantageous in that it supports cross-table referencing among multiple tables. It also allows reference editing during formula entry to allow convenient selection of other cells and capturing of their contents as variants used in the formula. The architecture further facilitates automatic universal recalculation throughout all tables in the document in response to user modification of a single table.

A document 120 is constructed and rendered on the document renderer 102. The document 120 combines one or more text-based body elements 122 with one or more tables 124. For discussion purposes, the document 120 is written in a markup language, such as XML (extensible markup language). XML documents have an advantage in that they can be transformed using XSL (extensible stylesheet language) and rendered directly as HTML (hypertext markup language). In this case, the renderer 102 may be implemented as a browser or other application that handles and renders HTML documents. The table 124 is thus rendered as an HTML table.

Fig. 2 shows an example document 120 that has body elements 122(1), 122(2), and 122(3) and a table 124 situated between body elements 122(2) and 122(3). In this example, the document 120 is a letter written to Mr. Jones describing various home improvement projects and the costs associated with the projects. In Fig. 2, the table 124 is in a non-editing mode and resembles a standard word processing table with three columns 202(1)-202(3) and five rows 204(1)-204(5).

Fig. 3 shows the same document 120 when the user is editing the table 124. Notice that table 124 now "looks" like a spreadsheet more than a traditional table. The table 124 has integrated column headers 302(1), 302(2), and 302(3), as well as

integrated row headers 304(1)-304(6). The table 124 has a column expansion control 306 and a row expansion control 308 to permit easy expansion of the table.

In this example, the user is entering a summation formula in cell C6. Using a mouse pointer 310, the user is referencing an array of cells C2 through C5 for entry into the formula. Upon confirmation (e.g., releasing the left mouse button), a reference to the cells C2-C5 are inserted into the summation formula in cell C6 and the formula is calculated to add the dollar amounts in column C. The result of $12,060 is inserted into cell C6. The many features of the table user interface are discussed in greater detail below under the section heading "User Interface Features".

With reference again to Fig. 1, the table and spreadsheet objects 104 and 106 provide editing functionality for the table 124, including such functions as table resizing, selection, cut, copy, paste, split, merge, table formatting, and a host of other rich spreadsheet events. The spreadsheet engine 112 provides the spreadsheet functionality for the table 124, including such functions as formula creation, reference editing, recalculation, and the like. Architecturally, the table and spreadsheet components are separate from one another, although the spreadsheet relies on the table and the table provides special notifications and events to help the spreadsheet. This allows either component to add additional functionality without directly affecting the other component.

The spreadsheet engine 112 includes a grid object 130 that receive events indicative of user activity in the table 124 and coordinates actions among various objects. There is one grid object 130 for each table created in the document 120. The workbook 110 tracks all grid objects 130 to resolve any cross-table referencing. Upon creation, the grid object 130 registers with the workbook 110

so that it can participate when tables are updated. The grid object keeps an interface to the spreadsheet objects 106 (this is technically a browser behavior but could be any object) to fetch values from the HTML tree maintained at the renderer 102.

The grid object 130 maintains two tables: a format table 132 and a cell table 134. The format table 132 holds information about the data format of each cell in the table 124. For instance, the cell may contain dates, numbers, dollar amounts, percentages, and so forth. The cell table 134 stores the actual data for each cell in the table 124. In the example shown in Fig. 3, the format table 132 would contain information that cells A1-A6, B1-B6, and C1 are text and cells C2-C5 are formatted as currency in U.S. dollars.

The cell table 134 holds the actual data in the cells of the table 124, such as text, values, and formulas. The cell table 134 stores pointers to multiple cells 136(1)-136(C), one for each cell in the table. Each cell 136 is an object with a variant containing the parsed value of the cell and a reference to complete information about the cell. If the cell contains text or numeric data (e.g., cells A1-A6, B1-B5, and C1-C5 in Fig. 3), it is stored directly in the variant. Formulas, such as the summation formula in cell C6 of Fig. 3, are stored as variants with a pointer to the appropriate formula object maintained by the formula manager 140 (discussed below).

The spreadsheet engine 112 includes a formula manager 140 to handle all formulas and parsing duties for formulas, data values, and references (e.g., D4:E23). The workbook 110 serves as the linkage between the formula manager 140 and the registered grids 130. The formula manager 140 maintains a recalculation engine 142 that performs recalculation of all formulas in response to

event changes in the table. In one implementation, the recalculation engine 142 maintains the formulas for a document in a bi-directional linked list, sometimes referred to as the "formula chain". Following a recalculation event (e.g., user entry of a new data value or new formula), the recalculation engine 142 traverses the list, evaluating formulas that may be affected by the event.

If the current formula depends on other formulas that have not yet been evaluated, the current formula is moved to the end of the list. After one recalculation pass, the formula list is organized in natural order and will not need to be reordered during subsequent recalculations unless new formulas are added. If recalculation comes to a formula that has already been bumped to the end of the list and discovers that this formula still relies on not-yet-calculated dependencies, the formula contains a circular reference. In this case, the recalculation engine returns a circular error.

The formula manager 140 also has a parser 144 that parses the formulas. In one implementation, the parser 144 is a recursive descent parser that extracts tokens from a stream and appends them to an array of character-size operation types and a parallel array of variant operands. When done, the parser 144 creates a new formula object 146 and gives it the two arrays of parsed information. The formula manager 140 therefore maintains one or more formula objects 146(1)-146(B) that contain formula information, including the parsed formula expression returned by the parser 144, the current result, the type of formula, and the current formula state.

The parser 144 is preferably a delay parser that parses cells only when necessary, such as the first time that a formula has been loaded or the first time a value has been edited or referenced. Most cells in the table, however, will not

contain a value that is referenced by a formula, so non-formula cells are only parsed as needed. If a cell is found to contain a formula when the table is loaded, the cell is parsed immediately and added to the recalculation chain. If the cell does not contain a formula, it is left unparsed until a formula requires its value.

In one implementation, there are three types of formulas: normal, semi-calculation, and non-calculation. The normal formula is reevaluated only when its dependencies change. The semi-calculation formula is reevaluated every time the recalculation engine 142 performs a recalculation operation. The non-calculation formula is never evaluated at all. Non-calculation formulas are a special formula type for handling nested tables (i.e., a table within a table) and free floating fields (i.e., a single table cell) that is nested within tables or other free floating fields.

Consider the case of an inner table nested inside a cell of an outer table. If the inner table contains a formula that changes to a different value following recalculation, the value of the outer table's cell will also change. Such a dependency is not encoded anywhere, since there is no formula in the outer table attached to the inner table. In such cases, a non-calculation formula is set in the outer table's cell to re-fetch the result value from the inner calculation. Thus, it participates in the normal dependency management of recalculation and all references to the outer table are updated when appropriate. Nested tables are described below in more detail.

In one implementation, the formula objects 146 are owned by a COM wrapper (not shown), which is in turn held onto by a cell object 136 in the grid 130 where the formula resides. The formula objects 146 are themselves part of the bi-directional linked list of formulas maintained by the recalculation engine 142. The formula objects 146 contain references to their home row and column and to

the cell object 136 in grid 130. The references allow the recalculation engine 142 to travel down the recalculation chain with formulas from several tables and easily determine to which table a given formula belongs. Many operations, from formula saving to table deletion, depend on this ability to traverse the chain.

The formula manager 140 also parses referenced cell groups. As examples, the formula manager 140 parses "A5" as a cell reference, "D4:E23" as a compound rectangular reference, "F30" as an absolute reference, "Table5!D5" as a cross-table reference, "Field3" as a whole-table cross-table reference, "A5:D5 B3:B6" as an intersection, and "D3,E4" as a union.

The non-core worksheet functions 114(1)-114(W) are optional elements. Examples of such functions include analysis functions, statistical functions, and trigonometric functions. The modular architecture 100 makes it flexible to remove unwanted worksheet functions or add new worksheet functions.

The spreadsheet object 106 is a counterpart to the grid object 130 located outside of the spreadsheet engine. There is one pair of a spreadsheet object 106 and a grid object 130 per table 124. The spreadsheet objects 106 define a behavior that receives events from the document renderer 102, processes them a little, and passes the events onto the grid object 130. In response to the events, the grid object 130 updates the per-table cell data in cell table 134 and/or formatting information in format table 132.

The spreadsheet behavior 106 has three objects: GridBehavior 150, CellEditing 152, and Spreadsheet 154. The GridBehavior object 150 provides a layer of abstraction between the grid object 130 and individual HTML table cells and allows the grid object 130 to access HTML values and styles. The GridBehavior object 150 wraps the HTML elements in a common interface so that

the grid 130 does not need to know the particular structure of the HTML table. Additionally, the GridBehavior object 150 manages table-specific portions of a "reference edit" operation.

The CellEditing object 152 and Spreadsheet object 154 interact directly with an HTML tree and the table behavior 104 to provide the grid 130 with events. The Spreadsheet object 154 is responsible for recording undo records for actions affecting the spreadsheet.

The CellEditing object 152 manages user-level editing of cells. It processes events related to user edits of in-cell data values and provides some editing user interface (UI) elements, including the formula edit box that permits user edits of formulas. When editing a formula, a floating formula edit box is provided above the cell's location and resized as necessary to accommodate the formula. The localized edit box eliminates a potential UI problem of forcing the user to stuff the entire formula into the table cell, which would cause the table (or paragraph) to resize strangely as the user brings up and dismisses the formula to be replaced by its result.

The CellEditing object 152 also supports the reference edit operation when the formula edit box is presented. As noted above, the reference edit operation allows the user to visually reference cells using a mouse pointer (or other focus mechanism) and in response, inserts a reference to that cell data in the current formula edit box. The formula edit box is described below in more detail. The CellEditing object 152 is only present when a cell is being actively edited.

The spreadsheet objects 106 handles top-level duties such as inserting a table or a free floating field and routing commands to the appropriate table based

on the current selection in the document 120. The spreadsheet objects 106 also creates and manages the workbook 110.

The integrated table and spreadsheet model eliminates the need for the user to choose the structure of data within a document prior to creating that document. Historically, if the user needed more control over the presentation of the tabular data, the user tended to select a word processing application. On the other hand, if the user required computations over the data, the user typically chose a spreadsheet application. The integrated architecture allows the user to combine several different types of data within one document.

Additionally, by integrating spreadsheet functionality inside a table, the user can build the document around the table. In spreadsheet applications, the user is restricted to the grid layout for all document content. In the integrated architecture, users may create a rich document that contains multiple tables, each with data that can be formatted as values and used in calculations throughout different tables.

Architecture with Free Floating Field

Fig. 4 shows an architecture 400 that is similar to that shown in Fig. 1, but illustrates how the architecture scales to accommodate multiple tables within a single document as well as free floating fields. The architecture 400 may be implemented at a standalone computer, a network server computer, a network client computer, or distributed at both the server and client.

In Fig. 4, the document 402 rendered by renderer 102 has multiple text-based body portions 404(1)-404(3), two tables 406(1) and 406(2), and one free floating field (FFF) 408. The free floating field 408 is akin to a spreadsheet value

that may be inserted anywhere in the document, including in the middle of a text-based body and appearing as a natural part of the text.

Figs. 5 and 6 show two examples of documents that have multiple tables and/or a table with a free floating field. In Fig. 5, a document 500 contains a first or outer table 502 and a second or inner table 504 nested within cell B3 of the outer table 502. The ability to nest tables is one feature of this architecture that conventional spreadsheet programs do not provide. In Fig. 6, a document 600 has three tables 602, 604, and 606, and a free floating field 608 that is currently being edited.

With reference again to Fig. 4, the spreadsheet engine has a grid object for each table and free floating field in the document 402, as represented by grid objects 130(1), 130(2), and 130(3). In addition, there is one spreadsheet behavior for each table 406 in the document 402, as represented by spreadsheet objects 106(1) and 106(2). The architecture 400 also has one free floating field behavior 410 for each free floating field 408 in the document. As a result, there is one pair of corresponding grid objects and spreadsheet/FFF behaviors for each table or free floating field in the document 402.

The grid object 130(2) used to support the free floating field object 410 is essentially the same as the grid objects 130(1) and 130(3) used to support the tables, which are described above in detail. One minor difference is that the grid object 130(2) contains only one cell object 136 because the free floating field 408 is similar to a table with only one cell.

The free floating field behavior 410 has three objects: an FFFBehavior object 412, a CellEditing object 414, and a Spreadsheet object 416. The CellEditing object 414 and Spreadsheet object 416 are identical to those in the

spreadsheet behavior 106, as described above with reference to Fig. 1. The FFFBehavior object 412 takes the place of the GridBehavior object in the context of free floating fields. Like the GridBehavior, the FFFBehavior object 412 provides an interface for the grid object 130(2) and manages "reference edit" operations for the free floating field.

Exemplary Computing Environment

Fig. 7 illustrates an example of an independent computing device 700 that can be used to implement the integrated spreadsheet/table architectures of Figs. 1 and 4. The computing device 700 may be implemented in many different ways, including a general-purpose computer (e.g., workstation, server, desktop computer, laptop computer, etc.), a handheld computing device (e.g., PDA, PIM, etc.), a portable communication device (e.g., cellular phone with computing capabilities), or other types of specialized appliances (e.g., set-top box, game console, etc.).

In the illustrated example, computing device 700 includes one or more processors or processing units 702, a system memory 704, and a bus 706 that couples the various system components including the system memory 704 to processors 702. The bus 706 represents one or more types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 704 includes read only memory (ROM) 708 and random access memory (RAM) 710. A basic input/output system (BIOS) 712, containing the basic routines that help to transfer information between elements within the computing device 700 is stored in ROM 708.

Computing device 700 further includes a hard drive 714 for reading from and writing to one or more hard disks (not shown). Some computing devices can include a magnetic disk drive 716 for reading from and writing to a removable magnetic disk 718, and an optical disk drive 720 for reading from or writing to a removable optical disk 722 such as a CD ROM or other optical media. The hard drive 714, magnetic disk drive 716, and optical disk drive 720 are connected to the bus 706 by a hard disk drive interface 724, a magnetic disk drive interface 726, and a optical drive interface 728, respectively. Alternatively, the hard drive 714, magnetic disk drive 716, and optical disk drive 720 can be connected to the bus 706 by a SCSI interface (not shown). It should be appreciated that other types of computer-readable media, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs), read only memories (ROMs), and the like, may also or alternatively be used in the exemplary operating environment.

A number of program modules may be stored on ROM 708, RAM 710, the hard disk 714, magnetic disk 718, or optical disk 722, including an operating system 730, one or more application programs 732, other program modules 734, and program data 736. As one example, the architecture 100 may be implemented as one or more programs 732 or program modules 734 that are stored in memory and executed by processing unit 702. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for computing device 700.

In some computing devices 700, a user might enter commands and information through input devices such as a keyboard 738 and a pointing device 740. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. In some instances, however, a computing

device might not have these types of input devices. These and other input devices are connected to the processing unit 702 through an interface 742 that is coupled to the bus 706. In some computing devices 700, a display 744 (e.g., monitor, LCD) might also be connected to the bus 706 via an interface, such as a video adapter 746. Some devices, however, do not have these types of display devices. Computing devices 700 might further include other peripheral output devices (not shown) such as speakers and printers.

Generally, the data processors of computing device 700 are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs or over the Internet. From there, they are installed or loaded into the secondary memory of a computing device 700. At execution, they are loaded at least partially into the computing device's primary electronic memory. The computing devices described herein include these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the steps described below in conjunction with a microprocessor or other data processor. The service system also includes the computing device itself when programmed according to the methods and techniques described below.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computing device 700, and are executed by the data processor(s) of the computer.

It is noted that the computer 700 may be connected to a network via a wire-based or wireless connection to interact with one or more remote computers. In this network context, the computer 700 may be configured to store and execute portions of the architecture 100, while one or more remote computers store and execute other portions of the architecture. For example, the document renderer 102 may reside on one computer, while the remaining components reside on a separate computer. As a result, the architecture is distributed, with various components being stored on different computer-readable media.

General Operation

Fig. 8 shows a table/spreadsheet process 800 implemented by the table/spreadsheet architecture 100 of Fig. 1. The process 800 may be embodied in software stored and executed on a computer, such as computing device 700 in Fig. 7. Accordingly, the process 800 may be implemented as computer-executable instructions that, when executed on a processing system such as processor unit 702, perform the operations and tasks illustrated as blocks in Fig. 8.

At block 802, the architecture 100 creates a corresponding set of table, spreadsheet, grid objects 104, 106, and 130 for a new table UI presented as part of the document. In one implementation, the GridBehavior object 150, CellEditing object 152, and Spreadsheet object 154 are initially created for the new table and then the Spreadsheet object 154 creates an associated grid object 130 in the spreadsheet engine 112. The grid object 130 includes a format table 132 and a cell table 134. If this is the first spreadsheet, a workbook 110 is also created. The grid 130 and workbook 110 then create other objects, including the formula manager 140 and cells 136 for each cell in the table being created.

At block 804, in response to the user entering data and/or formulas into the table, the architecture receives the user entry and passes it to the spreadsheet engine 112 for processing. More specifically, in the continuing exemplary implementation, the Spreadsheet object 154 receives a table-parsed notification from the document renderer 102 and passes it along to the grid 130 for the new table. Suppose, for example, the user creates the following table:

7	15
8	

The HTML code for this table is as follows:

```
<table>
  <tr><td>7</td><td FMLA="=A1+A2">15</td></tr>
  <tr><td>8</td><td></td></tr>
</table>
```

Using enumeration methods provided by the GridBehavior object 150, four cells 136(1)-136(4) are retrieved, one for each existing cell in the table. The spreadsheet object 154 receives a data value 7 for cell A1, a data value 8 for cell A2, and a formula for cell B1, and passes this information onto the spreadsheet engine 112.

At block 806, based on the user input, the architecture determines whether to parse the user-entered information or delay parsing until later. The architecture preferably employs a delay parser that parses cells when necessary, such as the first time that a formula has been loaded or the first time a value has been edited or

referenced. Most cells in the table, however, will not contain a value that is referenced by a formula, so non-formula cells are only parsed as needed. If a cell is found to contain a formula when the table is loaded, the cell is parsed immediately and added to the recalculation chain. If the cell does not contain a formula, it is left unparsed until a formula requires its value.

At block 808, assuming that parsing is needed now (i.e., the "no" branch from block 806), the architecture parses the user-entered information and updates the format table 132 and cell table 134 based upon this information. For example, cell A1 is parsed first, although the order is immaterial. The parser 144 evaluates whether a formula exists. In this case, no formula is found and the cell is left unparsed and marked to be parsed later. The corresponding 0,0 entry in cell table 134 is set to point to the unparsed cell 136(1).

Cell B1 is parsed next. Here, the parser 144 finds a formula "FMLA" attribute (i.e., "=A1+A2") and parses the formula string, returning the appropriate variant. The variant is placed in a new cell 136(2), which is stored in the cell table 134 at the appropriate location 0,1. Additionally, the formula is added to the chain of formulas maintained at the recalculation engine 142.

Cells A2 and B2 are parsed in a similar manner to A1 because neither cell contains a formula, resulting in references to unparsed cells 136(3) and 136(4) being added to the cell table 134. When all cells have been parsed, the recalculation engine initiates the first recalculation to determine actual values to be displayed in cells with formulas.

At block 810, the architecture determines whether recalculation is appropriate. Some user input may not require recalculation, such as insertion of a new data value that is not referenced by a formula. If recalculation is not needed,

flow continues at block 814 to determine whether the table needs to be updated in view of the user input.

At block 812, assuming recalculation is appropriate (i.e., the "yes" branch from block 810), the architecture recalculates the various formulas that may have been affected by the user input. In the ongoing example, the act of putting a value into a cell 136 in the cell table 134 triggers a data-changed notification to registered listeners, which includes the grid object 130. The grid object 130 identifies the changed cells and forwards the notification to the formula manager 140, which marks any formulas depending on the changed cells as dirty and in need of recalculation.

The grid object then calls the recalculation engine 142, which loops over the recalculation formula chain and recomputes the variously affected formulas (block 812(1)). While evaluating the formulas in the formula chain, unparsed cells that were previously left unparsed may now be parsed (block 812(2)). In this case, when the formula =A1+A2 is evaluated, the recalculation engine discovers that these cells are unparsed. It immediately asks the parser 144 to fully parse the cells. The parser 144 does so and enters values of 7 and 8 into the cell table 134. The recalculation engine can then evaluate the formula =A1+A2 using the new values that the parser 144 has found.

Once the recalculation engine 142 finishes, it returns control to the workbook 110. The workbook 110 calls back to the recalculation engine 142 to learn which formulas changed as a result of the recalculation cycle. The workbook 110 then locates the grid object 130 that holds the formula and calls it to save the cells that contain formulas whose results changed.

At block 814, the architecture determines whether any results have changed following the recalculation. If no results have changed (i.e., the "no" branch from block 814), process flow continues at block 804 for the next user input. Conversely, if the results have changed (i.e., the "yes" branch from block 814), the architecture 100 loads an updated table with the modified values and renders the updated table as part of the document (block 816).

It is noted that the above example assumed that the user entered data or a formula. The user may also change the format of one or more cells in the table. The process 800 handles format changes in essentially the same way, but accounts for those changes in the format table 132 rather than the cell table 134.

User Interface Features

The architecture 100/400 supports many user interface (UI) features in the rendered document to convey to the user that the table is not only a table, but also a full functioning spreadsheet. These UI features are described separately below, with reference to Figs. 2, 3, 5, and 6. Hand-in-hand with these features are the underlying operations and inner workings of various components in the architecture 100/400. These aspects will be described in more detail later in this disclosure under the heading "Functionality Features".

One of the primary benefits of integrating spreadsheet functionality into tables is that the user need no longer think in terms of whether the document should be primarily a spreadsheet document produced by a spreadsheet application (e.g., an ".xls" file from the "Excel" program) or primarily a word processing document produced by a word processing application (e.g., a ".doc" file from the "Word" program). Instead, the user creates an HTML document (or other markup-

based document, such as an XML document) that can have both text and spreadsheet/table components. By integrating spreadsheet functionality inside a table, the user can build the document around the table without being restricted to the grid layout for all document content, as in the case of spreadsheet programs.

Integrated Headers

The table 124 toggles between a "table look" (Fig. 2) and a "spreadsheet look" (Fig. 3) depending upon whether the user is editing the table. The spreadsheet look may be invoked in a number of ways, including by actively editing a cell, by hovering a pointer over the table, or by some other activity. As illustrated in Fig. 3, the spreadsheet look includes column headers 302(1)-302(3) and row headers 304(1)-304(6) that integrate with the table columns and rows, respectively. Visually, the column headers 302 appear just above the columns and are labeled with letters A, B, C, etc., where as the row headers 304 reside to the left of the rows and are labeled with numbers 1, 2, 3, etc.

Smart Selection

When the user selects a cell, the architecture intelligently discerns the type of content in the cell. For instance, the architecture determines whether the cell contains a data value, text, or a formula. If the selected cell contains text or a value, the UI exhibits the selection as a character-based cursor ready for cell editing. If the selected cell contains a formula, the UI exhibits the selection by highlighting the entire result of the formula. A second selection gesture will allow the user to edit the formula within the formula edit box.

Key Processing

Certain keys have different interpretations depending upon the contents of the cell. This dynamic interpretation accommodates the competing interests of a word processing table and a spreadsheet. As an example, the "Enter" key typically means return in word processing, whereas it means move to the next cell in a spreadsheet program.

If the cell contains text (e.g., cells A1-A6, B1-B5, and C1 in Fig. 3), the architecture interprets this cell as primarily being a word processing-based cell and treats the keys as if the user were working within a word processing application. Thus, an "Enter" key means return, a "tab" key means tab over some distance, the "=" key typed in anywhere but the beginning of the cell means the equals symbol without denoting a formula, and so forth.

If the cell contains a formula or a data value (e.g., cells C2-C6 in Fig. 3), the architecture interprets this cell as primarily being a spreadsheet-based cell and treats the keys as if the user were working within a spreadsheet application. Thus, an "Enter" key and "tab" key mean navigation commands to move to the next cell, the "=" key at the beginning of a cell implies the start of a formula, and so forth.

Table Expansion

Spreadsheet users are accustomed to the look and feel of an infinite grid. While the spreadsheet look of Fig. 3 does not have the same infinite grid feel, the table 124 has a column expansion control 306 and a row expansion control 308 that allow easy addition of columns and rows, respectively. The controls 306 and 308 include an actuatable addition icon together with a dashed column/row to

suggest that additional columns and rows may be added by simply actuating the icon.

Resize Behavior

The table 124 preserves column width and wraps text as the user enters sentences and phrases. In Fig. 3, notice that item 2 in cell B5 wraps within the cell, rather than resizing the column width to accommodate the entire item.

Formula Edit Box

The architecture provides a formula edit box for inserting or editing a formula in a table cell or free floating field. The formula edit box overlays the original cell in which the formula resides and initially defaults to the size and shape of the cell. If the formula exceeds the initial size, the formula edit box is resized to accommodate the formula. During resizing, the formula edit box initially grows horizontally in length, and then vertically in depth. The underlying table does not resize. The ability to resize the local formula edit box, rather than the cell and the table, eliminates a potential UI problem of watching the table resize strangely as the user clicks into and out of the cell containing the formula.

Examples of the formula edit box are illustrated in Figs. 3, 5 and 6. In Fig. 3, a formula edit box 312 hovers over cell C6 to accept the summation formula. In Fig. 5, a formula edit box 506 floats above cell C3 and is resized to accommodate the expanded formula. Notice that the underlying table cell C3 is not resized, but remains the same size and shape within the table. In Fig. 6, a formula edit box 610 resides above the free floating field 608 and is resized to hold the long formula.

Reference Edit

The table 124 allows the user to perform a reference edit operation, in which the user references one or more cells to extract their data values for inclusion in a formula in another cell. In Fig. 3, the user begins entering a summation formula (i.e., "=SUM(") in the formula edit box 312 above cell C6. The user then references cells C2 through C5 using a pointer 310 or some other mechanism. The referenced cells C2:C5 are highlighted or otherwise indicated as being selected, as represented by the bold rectangular box around the cells.

When the user finishes selecting the desired cells (e.g., releasing the left mouse button after highlighting cells C2:C5), the referenced cells are added to the summation formula in formula edit bock 312 (i.e., "=SUM(C2:C5)"). Upon further confirmation by the user (e.g., pressing the "Enter" key), an update event is generated and the architecture 100 recalculates the formula and updates the cell C6 to display the sum of the cells C2:C5, or $12,060, in cell C6.

Cross-Table Referencing and Universal Recalculation

The architecture 100 supports cross-table references where a cell in one table contains a formula referencing a cell in another table. The architecture 100 also supports cross-table reference edit operations that permit a user to reference a cell in one table or a free floating field when entering a formula into another table.

Fig. 6 illustrates cross-table referencing, where table 606 contains references to tables 602 and 604. All three tables are separate and independent from one another, and architecturally have their own set of grid, spreadsheet, table objects 130, 106, and 104. In Fig. 6, cell B2 in table 606 contains a summation formula for adding values in cells B2 and B3 of table 602 (i.e.,

=SUM(Table1!B2:Table1!B3)). Cell B3 in table 606 contains a summation formula for adding values in cells B2 through B4 in table 604 (i.e., =SUM(Table2!B2:Table2!B4)).

The ability to cross-reference other tables or free floating fields is beneficial in that all tables and free floating fields can be universally updated for any change in just one of the tables. For example, suppose the user changes the value in cell B2 of table 602 from $300 to $400. As a result of this change, table 602 is updated to reflect the new value $400 and the total amount in cell B6 is updated from $3,060 to $3,160. Additionally, the value in cell B2 of table 606 is updated to $2,400, causing the total amount in cell B4 in table 606 to be changed from $7,800 to $7,900.

The cross-table referencing is a significant improvement over conventional OLE techniques of embedding a spreadsheet object within a word processing document. With OLE, each spreadsheet object is independent of another and cannot reference cells in one another automatically. Since there is no cross-referencing ability, the OLE approach cannot support universal updating throughout the document's spreadsheets as a result of changing a value in one spreadsheet.

Free Floating Field Reference Edit

The reference edit operation is also available when entering a formula for a free floating field. Consider the document 600 in Fig. 6. The writer is attempting to summarize the total cost of all work items in the opening paragraph. Rather than typing in a hard value, the user decides to insert a free floating field 608 that

will hold the total for the job. By using a free floating field 608, the amount can be automatically updated as other estimates in the underlying tables are modified.

Using a reference edit operation, the user can enter the formula in the edit box 610 for free floating field 608 by selecting cell B6 in table 602 to capture element "Table1!B6" and then selecting cell B8 in table 604 to capture element "Table2!B8". When the user confirms this formula, the formula edit box 610 disappears and the total value of "$12,060" is inserted into the free floating field 608.

In the event the user subsequently changes the estimate of any item in tables 602 or 604, the total value in free floating field 608 is automatically updated. Extending a previous example, suppose the user changes the value in cell B2 of table 602 from $300 to $400. As a result of this change, table 602 is updated to reflect the new value $400 and the total amount in cell B6 is updated from $3,060 to $3,160. The value in cell B2 of table 606 is updated to $2,400, causing the total amount in cell B4 in table 606 to be changed from $7,800 to $7,900. Additionally, the total amount in free floating field 608 is updated from $12,060 to $12,160. All of the updating throughout the tables and free floating fields is performed automatically in response to the user's change of a single cell in a single table.

It is further noted that a free floating field may reference another free floating field. For instance, another free floating field may be added in document 600 to reference the first free floating field 608, or a combination of the free floating field 608 and a table cell in tables 602, 604, and 606.

Nested Table

The architecture 100 supports tables nested within one another. Fig. 5 illustrates a situation in which an inner table 504 is nested within a cell B3 of outer table 502. The two tables are independently managed and each has its own underlying set of grid object 130, spreadsheet object 106, and table object 104. In this example, cell C3 in outer table 502 is referencing an array of cells B1:B3 in inner table 504, as represented by the summation formula in formula edit box 506. Notice that the reference syntax "Table2!B1" in the formula edit box refers to a separate table and not to cell B3. This is essentially the same cross-table reference edit operation described above, even though the reference is to a table nested within another table cell.

The nested table is another feature that is an improvement over conventional table and spreadsheet approaches. OLE mechanisms of embedding a spreadsheet object within a word processing document do not support nested tables.

Common Document Behaviors

The architecture allows common document behaviors for the text body and across the tables. Such functions as spell checking, grammar checking, find, and replace are continuous across table boundaries, treating the cell contents as if they were part of the document. Moreover, text formatting carries across boundaries. Essentially, any features that are added to modify the text are similarly applied across a table boundary to text inside a table. The conventional OLE mechanisms of embedding a spreadsheet object within a word processing document were

incapable of supporting these common document behaviors that traversed table boundaries.

Functionality Features

This section describes how the architecture functionally supports the user interface features described above, including recalculation, reference edit mechanics, cross-table referencing, the formula edit box, and structure changes to the table. These functionality features are described separately below.

Data v. Presentation

The integrated table/spreadsheet architecture 100/400 separates data functions from presentation functions of the integrated table/spreadsheet by employing dual objects per table or floating field. As shown in Fig. 4, there is one pair of spreadsheet and grid objects for each table or floating field. The grid object 130 maintains the data and format information, and facilitates the recalculation process. The corresponding spreadsheet objects 106 are more concerned with presenting the table and free floating field as part of the document, as well as capturing user inputs into the table and free floating field.

The separation is beneficial because it allows the architecture to support cross-table referencing, reference editing to other tables, and universal recalculation throughout the document. The underlying grid objects do not care how the tables are laid out or where they appear in the document, nor do they care if there are one or multiple grid objects. Similarly, the spreadsheet objects 106 do not worry about the data and formulas, or the recalculation process.

Recalculation

Recalculation is a function performed by the architecture 100 in response to modification of a table or free floating field. When a modification is made, the architecture 100 recalculates the various formulas in all tables and free floating fields in the document that may have been affected by user input.

Continuing the example of Fig. 6, when the user changes the value in cell B2 of first table 602 from $300 to $400, the recalculation engine 142 in spreadsheet engine 112 recalculates the formulas in cell B6 of first table 602 to update the amount from $3,060 to $3,160. The recalculation engine 142 also re-computes the formula in cell B2 of third table 606 to yield $2,400 and the formula in cell B4 in third table 606 to yield $7,900. Finally, the recalculation engine 142 recalculates the formula in free floating field 608 to update the value from $12,060 to $12,160. This recalculation occurs automatically across the entire document in response to the user input.

Fig. 9 illustrates the recalculation process 900 for a single table in more detail. An input version of the user interface table 902(1) is shown at a time when the user enters a new value "7" in cell A1, but has not yet confirmed the entry (e.g., by hitting the "Enter" key or clicking out of the cell). An output version of the UI table 902(2) is shown at a time just after user confirmation.

A corresponding pair of spreadsheet and grid objects 106 and 130 exists for the table 902. The grid object 130 maintains a cell table 134 and a format table 132. Prior to user entry of "7" into cell A1, cell table 134 contains a value "1" in cell A3, a formula referencing cell A1 (i.e., "=A1") in cell C1, and a formula summing cells A1 and C1 (i.e., "=A1+C1") in cell C3. The format table 132 indicates that cell A3 is formatted as a number, and that cells C1 and C3 are

formatted as currency in U.S. dollars. The input version of UI table 902(1) shows results of the formatted formulas as $0.00 in cell C1 and $1.00 in cell C3.

Now, suppose the user enters the value "7" into cell A1 of UI table 902(1), as indicated by the pointer 904. The value is received at the spreadsheet objects 106, as indicated by flow arrow 910. Once the user confirms this entry by moving the selection out of the cell A1, the newly entered value "7" is passed to the spreadsheet engine 112 and particularly, the parser 144 of formula manager 140 (flow arrow 912).

The parser 144 parses the entry and determines it to be a data value. The parser 144 puts the data value into cell A1 of the cell table 134 (flow arrow 914). This insertion causes a table change event, which is sent to the recalculation engine 142 to initiate a recalculation (flow arrow 916). The recalculation engine 142 runs through the formula chain to recalculate any formula anywhere that is affected by the new data value in cell A1. In this case, the formulas in cells C1 and C3 are affected and hence, these formulas are recalculated (flow arrow 918). The recalculation produces a result of "7" in cell C1 and a result of "8" in cell C3.

Afterwards, the format table 132 is consulted to determine the desired format for the new value and recalculated formulas (flow arrow 920). Here, the formula results are formatted as currency as indicated by the "$" symbols in cells C1 and C3, and the new value "7" is formatted as a number as indicated by the "#" symbol in cell A1.

The spreadsheet engine returns the formatted results $7.00 and $8.00 to the spreadsheet objects 106 (flow arrow 922). The spreadsheet objects 106 updates the table with these formatted results to produce the output version of the UI table 902(2) (flow arrow 924).

The recalculation event is essentially instantaneous. The user merely sees an immediate change in the UI table from input version 902(1) to output version 902(2).

Reference Edit Mechanics

The reference edit mechanism allows the user to reference another cell to obtain data, rather than forcing the user to type in a value or the reference syntax. In Fig. 9, consider the situation when the user created the formula "=A1" in cell C1. The user selects cell C1, types in an "=" sign to indicate a formula, and then references the cell A1 by moving the mouse pointer 904 to cell A1 and clicking. The spreadsheet objects 106 (namely, CellEditing object 152) capture this reference and pass it to parser 144. The parser 144 recognizes it as a formula, creates a formula object and inserts the formula into a cell of cell table 134, as indicated by cell C1.

Cross-Table Referencing and Universal Recalculation

With architecture 100, reference editing may be extended across multiple tables and free floating fields distributed throughout a document. A cell in one table or a free floating field may reference a cell in another table, a different free floating field, or a combination of a table cell and free floating field. The architecture 100 automatically recalculates all tables and free floating fields that are affected by a change in any one table cell or free floating field.

Fig. 10 illustrates the recalculation process 1000 for a document 1002 containing multiple tables 1004(1),..., 1004(N) and multiple free floating fields 1006(1), ..., 1006(M) distributed throughout the text body. A corresponding pair

of spreadsheet and grid objects 106 and 130 is created for each table 1004(1)-1004(N) and each free floating field 1006(1)-1006(M) in the document 1002.

In this illustration, spreadsheet object 106(1) and associated grid object 130(1) support UI table 1004(1), spreadsheet object 106(N) and associated grid object 130(N) support UI table 1004(N), spreadsheet object 106(N+1) and associated grid object 130(N+1) support free floating field 1006(1), and spreadsheet object 106(N+M) and associated grid object 130(N+M) support free floating field 1006(M). The table grid objects 130(1)-130(N) each contain a cell table 134(1)-134(N) and a format table 132(1)-132(N). The FFF grid objects 130(N+1)-130(N+M) each contain a single cell 136(1)-136(M) and a corresponding format cell 138(1)-138(M).

Suppose the user is entering a summation formula in cell B1 of UI table 1004(N) that adds three cells C1-C3 in table 1004(1). Rather than typing in the reference syntax (i.e., "=SUM(Table1!C1:Table1!C3)"), the user may simply move the pointer 1010 to table 1004(1) and select the desired array of cells, as indicated by the selection block 1012. The spreadsheet object 106(N) (namely, the CellEditing object) associated with the source table 1004(N) recognizes the reference edit operation and captures the selected cells C1-C3 in remote referenced table 1 (flow arrow 1020). When the user confirms this entry by moving the selection out of the referenced table 1004(1), the newly entered formula "=SUM(Table1!C1:Table1!C3)" is passed to the spreadsheet engine 112 and particularly, the parser 144 of formula manager 140 (flow arrow 1022).

The parser 144 determines that the entry is a formula and creates a formula object (not shown) and adds the formula to the formula chain. The parser 144 puts the formula into cell B1 of the cell table 134(N) in cell table N (flow arrow 1024).

This insertion generates a table change event, which is sent to the recalculation engine 142 to initiate a recalculation (flow arrow 1026).

The recalculation engine 142 runs through the entire formula chain to recalculate any formula in any table or free floating field that is affected by adding the new formula in cell B1. In this case, the formulas in free floating cells 136(1) and 136(M) are affected. But, since these formulas rely on the result of the newly entered formula in table N, they are moved to the end of the formula chain. Thus, the new formula is first calculated (flow arrow 1028), and then the formulas in free floating field cells 136(1) and 136(M) are recalculated (flow arrows 1030 and 1032). The recalculation produces a result of "425" for table cell B1 in cell table N, a result of "425" in FFF cell 136(1), and a result of "425" in FFF cell 136(M).

Afterwards, the various format tables 132(N) and format cells 138(1) and 138(M) are consulted to determine the desired format for the results of the recalculated formulas. Here, all formula results are formatted as currency as indicated by the "$". The spreadsheet engine 112 returns the formatted results "$425.00" to the associated spreadsheet objects 106(N), 106(N+1), and 106(N+M) (flow arrows 1034, 1036, 1038). The spreadsheet objects then update their associated table and free floating fields with these formatted results to produce the output as shown in Fig. 10 (flow arrows 1040, 1042, 1044).

Once again, the recalculation event is essentially instantaneous and the user merely perceives an immediate change in the affected UI table and free floating fields throughout the document 1002.

Formula Edit Box

The CellEditing object 152 manages the formula edit box that permits user edits of formulas. The formula edit box is provided in the user interface in response to the user entering the "=" symbol at the beginning of a cell. The formula edit box is overlaid as a separate entry field above the cell into which the user is inserting a formula. The CellEditing object 152 captures the user entry of various variants in the formula, and facilitates reference editing to other cells as a way to input variants. When the formula is entered via the formula edit box and confirmed, the CellEditing object 152 passes the formula to the formula manager 140 in the spreadsheet engine 112 for parsing. The formula edit box is then removed from the user interface.

Structural Changes

The Table object 104 manages and monitors the user input for structure changes, such as insertion/deletion of a row, merging cells, and so forth. When the user makes a structure change, the Table object 104 fires events to the GridBehavior object 150, which informs the spreadsheet engine 112, and in turn updates the cell table 134 associated with the UI table.

As above, any change to the cell table causes an event that is returned to the recalculation engine 142 to initiate a recalculation cycle. The recalculation engine steps through the chain of formulas and updates all cells affected by the structural change, returning any errors that might arise from the structure change (e.g., loss of a reference value as a result of deleting a row/column). The spreadsheet engine then outputs the results of the recalculation and the UI table is updated to reflect the structural change and the recalculation results.

Cut, Copy, Paste

A separate document object may be employed to manage operations affecting the entire document, rather than a specific table. The document object plays a role in initially inserting the table/spreadsheet into the document. The document object may be constructed similarly to the GridBehavior object 150 and configured to monitor for cut, copy, and paste operations. When a cut or copy operation is performed, the object places the HTML code on the clipboard. Upon a subsequent paste operation, the HTML code is retrieved from the clipboard and inserted into the appropriate location. It is noted that, unlike some spreadsheet programs, the user is not forced to cut/copy and then *immediately* paste.

One unique problem that is not encountered by traditional spreadsheet programs is the ability to create a new table through a paste operation. When a new table is created, the architecture automatically renames the new table and adjusts all references within the table that were pasted.

Conclusion

Although the description above uses language that is specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the invention.